Heuristic Search

Beam Search

- Limit open list to k most promising nodes
- k is the beam width
- “Promising” nodes can be determined by f, h, other techniques
- Memory is bounded
- Solutions are not guaranteed to be optimal

Weighted A*

- $f(n) = g(n) + w * h(n)$
- When $w = 1$, this is just A*
- When $w > 1$, node expansion preference is given to nodes that seem closer to a goal
- Solutions are not guaranteed to be optimal

Other Shortest Path Algorithms

- IDA*
- RBFS
- SMA*
- RTA*, LRTA*

Constraint Satisfaction Problems

Other Search Problems

Map coloring: Given a map of n regions and a set of k colors, color every region differently from its neighbors

n-queens: Given an $n \times n$ chess board, arrange n queens so that none is threatening another

Sudoku: Fill in digits on a Sudoku board without breaking any rules

What algorithm can we use to solve these?

Types of Search Problems

- Shortest Path (SampleWorld, tile puzzle, driving directions)
  - given operators and costs
  - want least-cost path to goal
  - goal depth/cost is unknown
- Constraint Satisfaction
  - any goal is fine
  - fixed depth
  - explicit constraints

CSP Definition

Each CSP is defined by components X, D, C:

- X: a set of variables $X_1 \ldots X_n$
- D: a set of domains $D_1 \ldots D_n$, corresponding to the allowed values of n variables
- C: a set of constraints that describe restrictions or required relationships between variables

The solution to a CSP is a set of values corresponding to each variable: $X_1 = v_1 \ldots X_n = v_n$

Chronological Backtracking

Do not expand any node in search tree that violates a constraint

Forward Checking

When assigning a variable, remove conflicting values for all connected variables
Backtrack if a variable has no possibilities left

Arc consistency: for every value in domain of variable x, there exists a value in domain of y that satisfies all constraints

Heuristics for CSPs

Variable choice: choose variable with most constraints (smallest domain)
Keep search tree small

Value choice: choose variable with least constraints (fewest removals)
Find valid solution sooner

Maintaining Arc Consistency

Ensure every value for x has a legal value in all neighbors y.

If one doesn't, remove it and ensure consistency of all y.

Maintaining Arc Consistency

AC-3(csp)

```
Q ← all arcs in csp
while Q is not empty do
    (x, y) ← pop Q
    if REVISED(x, y) then
        if x's domain is empty, return failure
        for every other neighbor z of x do
            push (z, x) on Q
```

REVISE(csp, x, y)

```
revised ← false
for each v in domain of x do
    if no value in y is compatible with v then
        remove v from x's domain
        revised ← true
return revised
```

Other CSP Algorithms

- Backjumping
- Dynamic Backtracking
- Randomized Restarts